DISTINCT: i dlaczego zwykle jest objawem, nie rozwiązaniem

> `DISTINCT` usuwa zduplikowane wiersze z wyniku, ale w zapytaniu ze złączeniem duplikaty najczęściej biorą się z błędu w samym złączeniu — a wtedy naprawiamy złączenie, zamiast doklejać `DISTINCT` jako plaster.

Skróty użyte na tej stronie (2)

Skrót	Co znaczy
ISO	International Organization for Standardization: organizacja, która wydaje normy techniczne używane na całym świecie
SQL	Structured Query Language: język zapytań do relacyjnych baz danych

Fiszka

TYLKO ODCZYT

Usunięcie duplikatów z jednej kolumny

```
SELECT DISTINCT status FROM zamowienia;
```

`DISTINCT` porównuje kompletne wiersze wyniku i zostawia po jednym egzemplarzu z każdej grupy identycznych. Osiem zamówień ma tylko cztery różne statusy — reszta to powtórzenia.

Przykład:

```
    status
--------------
 zrealizowane
 anulowane
 w realizacji
 nowe
(4 rows)
```

TYLKO ODCZYT

Usunięcie duplikatów po wielu kolumnach

```
SELECT DISTINCT klient_id, status FROM zamowienia ORDER BY klient_id, status;
```

Przy kilku kolumnach za duplikat uznaje się dopiero identyczną kombinację wszystkich wymienionych wartości naraz. Zamówienia 101 i 102 mają tę samą parę `(klient_id, status)` — `(1, zrealizowane)` — więc kolapsują do jednego wiersza; pozostałe kombinacje są unikalne już same z siebie.

Przykład:

```
 klient_id |    status
-----------+--------------
           | nowe
         1 | zrealizowane
         2 | anulowane
         2 | nowe
         3 | w realizacji
         4 | zrealizowane
         5 | anulowane
(7 rows)
```

TYLKO ODCZYT

Pierwszy wiersz z każdej grupy: DISTINCT ON

```
SELECT DISTINCT ON (klient_id) klient_id, id, data, wartosc
FROM zamowienia
```

```
WHERE klient_id IS NOT NULL
ORDER BY klient_id, data DESC;
```

Rozszerzenie właściwe PostgreSQL. `DISTINCT ON (klient_id)` zostawia dokładnie jeden wiersz dla każdej wartości `klient_id` — ten, który jako pierwszy pojawia się w kolejności zadanej przez `ORDER BY`. Tu `ORDER BY klient_id, data DESC` sprawia, że dla każdego klienta zostaje jego najnowsze zamówienie.

Przykład:

```
 klient_id | id |    data    | wartosc
-----------+----+------------+---------
         1 | 102| 2024-02-03 |  430.50
         2 | 105| 2024-03-11 |  150.00
         3 | 104| 2024-03-10 | 2890.00
         4 | 107| 2024-04-02 | 3200.00
         5 | 108| 2024-04-05 |    0.00
(5 rows)
```

Kolumna, po której grupujemy w `DISTINCT ON`, musi być pierwsza w `ORDER BY` — inaczej PostgreSQL zgłosi błąd, że wyrażenia `DISTINCT ON` i `ORDER BY` nie są zgodne.

□□ TYLKO ODCZYT

Deduplikacja przez GROUP BY zamiast DISTINCT

```
SELECT klient_id, status FROM zamowienia GROUP BY klient_id, status ORDER BY klient_id,
status;
```

`GROUP BY` bez żadnej funkcji agregującej daje dokładnie ten sam wynik co `DISTINCT` na tych samych kolumnach — obie konstrukcje zostawiają po jednym wierszu z każdej unikalnej kombinacji wartości. Wybór między nimi bywa kwestią czytelności: `GROUP BY` czytelniej zapowiada, że w tym samym miejscu może się pojawić agregat.

Przykład:

```
 klient_id |    status
-----------+--------------
           | nowe
         1 | zrealizowane
         2 | anulowane
```

```
          2 | nowe
          3 | w realizacji
          4 | zrealizowane
          5 | anulowane
(7 rows)
```

🔒 TYLKO ODCZYT

Sprawdzenie istnienia zamiast złączenia: EXISTS

```
SELECT nazwa, miasto
FROM klienci k
WHERE EXISTS (
    SELECT 1 FROM zamowienia z WHERE z.klient_id = k.id
);
```

Zakładamy, że do tabeli `klienci` dopisano szóstego klienta, Górski Serwis z Katowic, który nie złożył jeszcze żadnego zamówienia. `EXISTS` sprawdza dla każdego klienta, czy istnieje choć jedno pasujące zamówienie, i zwraca wiersz klienta dokładnie raz — niezależnie od tego, ile zamówień faktycznie ma. W odróżnieniu od złączenia z `zamowienia` nie ma tu mowy o duplikacji ani o potrzebie sprzątania jej przez `DISTINCT`, bo `EXISTS` nigdy nie wprowadza do wyniku kolumn z podzapytania.

Przykład:

```
           nazwa               |  miasto
-------------------------------+-----------
 Nowak i Wspólnicy Sp. z o.o.  | Warszawa
 Kowalski Handel               | Kraków
 Zielona Dolina S.A.           | Wrocław
 Baltic Trade Sp. z o.o.       | Gdańsk
 Podlaski Handel               | Białystok
(5 rows)
```

Górski Serwis nie pojawia się w wyniku — nie dlatego, że ktoś go odfiltrował `DISTINCT`em, tylko dlatego, że podzapytanie `EXISTS` dla niego nie znajduje żadnego wiersza.

🔒 TYLKO ODCZYT

Deduplikacja funkcją okna: ROW_NUMBER

```
SELECT klient_id, id, data, wartosc
FROM (
    SELECT klient_id, id, data, wartosc,
           ROW_NUMBER() OVER (PARTITION BY klient_id ORDER BY data DESC) AS
numer_w_grupie
    FROM zamowienia
    WHERE klient_id IS NOT NULL
) AS ponumerowane
WHERE numer_w_grupie = 1
ORDER BY klient_id;
```

Ten sam efekt co `DISTINCT ON`, ale przenośny między silnikami — `ROW_NUMBER()` jest częścią standardu i działa tak samo w PostgreSQL, SQL Server, Oracle i MySQL od wersji 8. Funkcje okna mają własną stronę, Funkcje okna — OVER, PARTITION BY, ROW_NUMBER, RANK, LAG/LEAD, bo mechanizm `PARTITION BY` wykracza poza temat tej strony.

Przykład:

```
 klient_id | id |    data    | wartosc
-----------+----+------------+---------
         1 | 102| 2024-02-03 |  430.50
         2 | 105| 2024-03-11 |  150.00
         3 | 104| 2024-03-10 | 2890.00
         4 | 107| 2024-04-02 | 3200.00
         5 | 108| 2024-04-05 |    0.00
(5 rows)
```

> **Uważaj**
>
> - `SELECT DISTINCT *` na szerokiej tabeli porównuje każdą kolumnę każdego wiersza z każdym innym — koszt rośnie z liczbą kolumn, nie tylko z liczbą wierszy.
> - `ORDER BY` przy `DISTINCT` musi sortować po czymś, co jest w liście `SELECT` — inaczej silnik zgłasza błąd, opisany w sekcji „Częste błędy”.

□□ **SQL** = Structured Query Language: język zapytań do relacyjnych baz danych ·**LAG** = Link Aggregation Group: grupa łączy spięta w jedno logiczne; wariant MC-LAG rozciąga ją na dwa przełączniki

Po co to istnieje

> **Pojęcie: DISTINCT.** Deduplikacja wyniku zapytania — usunięcie z wyniku wierszy, które po policzeniu wszystkich kolumn z listy `SELECT` są identyczne z którymś wcześniejszym wierszem. `DISTINCT` działa na gotowym wyniku, dokładnie w kroku szóstym logicznej kolejności wykonania opisanej na stronie SELECT: kolejność zapisu a kolejność wykonania — po tym, jak `SELECT` policzył wszystkie kolumny i wyrażenia wynikowe, a przed sortowaniem przez `ORDER BY`.

Jak to działa

Rozwiń: jak to działa pod spodem

Skąd biorą się duplikaty w zapytaniu ze złączeniem. Złączenie tabeli nadrzędnej z tabelą podrzędną, w której jednemu wierszowi nadrzędnemu odpowiada wiele wierszy podrzędnych, mnoży wiersz nadrzędny tyle razy, ile ma dopasowań — mechanizm ten szczegółowo opisuje strona Czym jest złączenie: iloczyn kartezjański i warunek, który go ogranicza. Zapytanie, które łączy `zamowienia` z `pozycje` tylko po to, żeby pokazać kolumny samego zamówienia, produkuje duplikat zamówienia za każdą pozycję, jaką to zamówienie ma.

```
SELECT z.id, z.status, z.wartosc
FROM zamowienia z
JOIN pozycje poz ON poz.zamowienie_id = z.id
ORDER BY z.id;
```

Przykład:

```
 id |    status     | wartosc
----+---------------+---------
101 | zrealizowane  | 1250.00
101 | zrealizowane  | 1250.00
102 | zrealizowane  |  430.50
102 | zrealizowane  |  430.50
104 | w realizacji  | 2890.00
104 | w realizacji  | 2890.00
```

```
107 | zrealizowane  | 3200.00
107 | zrealizowane  | 3200.00
(8 rows)
```

Zamówienia 101, 102, 104 i 107 mają po dwie pozycje, więc każde z nich pojawia się dwukrotnie. Doklejenie `DISTINCT` do tego zapytania rzeczywiście usunie widoczne powtórzenia:

```
SELECT DISTINCT z.id, z.status, z.wartosc
FROM zamowienia z
JOIN pozycje poz ON poz.zamowienie_id = z.id
ORDER BY z.id;
```

Przykład:

```
 id |    status     | wartosc
----+---------------+---------
101 | zrealizowane  | 1250.00
102 | zrealizowane  |  430.50
104 | w realizacji  | 2890.00
107 | zrealizowane  | 3200.00
(4 rows)
```

Wygląda na naprawiony wynik, ale zamówienia 103, 105, 106 i 108 zniknęły z niego całkowicie — nie dlatego, że nie istnieją, tylko dlatego, że złączenie z `pozycje` jest złączeniem wewnętrznym, a te cztery zamówienia nie mają żadnej pozycji, więc nigdy nie trafiają do wyniku złączenia. `DISTINCT` naprawił objaw, który było widać — powtórzone wiersze — i ukrył objaw, którego widać nie było — brakujące wiersze. Prawdziwa naprawa nie potrzebuje `DISTINCT` w ogóle, bo zapytanie w ogóle nie potrzebowało `pozycje`:

```
SELECT id, status, wartosc FROM zamowienia ORDER BY id;
```

Ta wersja zwraca poprawnie wszystkie osiem zamówień, bez duplikatów i bez ukrytych braków, bo nigdy nie łączyła się z tabelą, której kolumn nawet nie używała w wyniku. Gdyby zapytanie rzeczywiście potrzebowało wiedzieć, czy zamówienie ma jakąkolwiek pozycję, właściwym narzędziem jest `EXISTS` z Fiszki tej strony, nie złączenie doprawione `DISTINCT`em.

Koszt DISTINCT. Żeby znaleźć duplikaty, silnik musi porównać każdy wiersz z każdym innym — w praktyce robi to, sortując cały wynik (żeby identyczne wiersze znalazły się obok siebie) albo budując dla niego tabelę haszującą. Obie metody kosztują dodatkową pracę proporcjonalną do rozmiaru całego wyniku, widoczną w planie wykonania jako krok `Sort` albo `HashAggregate` — strona Plan wykonania: jak go wywołać i jak go czytać pokazuje, jak ten krok

rozpoznać.

Kiedy DISTINCT jest właściwym narzędziem. Wtedy, gdy zapytanie rzeczywiście pyta o zbiór unikalnych wartości, a nie maskuje efekt uboczny złączenia — pierwsza Fiszka tej strony, lista unikalnych statusów zamówień bez żadnego złączenia w tle, jest dokładnie takim przypadkiem. `DISTINCT` jest też uzasadniony przy łączeniu wielu źródeł przez `UNION` (który domyślnie i tak usuwa duplikaty), przy raportach agregujących dane z wielu systemów, gdzie te same rekordy mogły trafić do zapytania dwoma różnymi drogami, albo po prostu przy eksploracyjnym zapytaniu w konsoli, gdzie liczy się szybka odpowiedź, nie plan wykonania.

Częste błędy

Doklejanie DISTINCT odruchowo, gdy wynik ma „za dużo” wierszy. Zamiast sprawdzić, skąd biorą się powtórzenia — czy złączenie ma właściwy warunek, czy tabela podrzędna faktycznie powinna brać udział w zapytaniu — `DISTINCT` ląduje na końcu listy `SELECT` jako pierwszy odruch. Efekt widać w sekcji „Jak to działa”: symptom znika, przyczyna zostaje, a dodatkowo znikają wiersze, które nie powinny były zniknąć.

ORDER BY po kolumnie spoza SELECT DISTINCT.

```
SELECT DISTINCT status FROM zamowienia ORDER BY wartosc;
```

Przykład:

```
ERROR:  for SELECT DISTINCT, ORDER BY expressions must appear in select list
LINE 1: SELECT DISTINCT status FROM zamowienia ORDER BY wartosc;
                                                        ^
```

`DISTINCT` deduplikuje po kolumnach z `SELECT` — `wartosc` w tym zapytaniu w ogóle nie istnieje w wyniku po deduplikacji, więc nie ma po czym sortować. Poprawka to dopisanie `wartosc` do listy `SELECT` albo usunięcie jej z `ORDER BY`.

SELECT DISTINCT * na tabeli z kolumną tekstową o zmiennej długości bez potrzeby. Porównanie całych, długich wartości tekstowych kolumna po kolumnie dla każdej pary wierszy jest dużo droższe niż porównanie samych kluczy — jeśli deduplikacja ma sens tylko względem części kolumn, `DISTINCT ON` albo `GROUP BY` na tej węższej liście kosztuje mniej.

□□ Ćwiczenia

> Zadania sprawdzające.

Ćwiczenie 1 — znalezienie przyczyny zamiast objawu

Zapytanie `SELECT DISTINCT p.nazwa, p.kategoria FROM produkty p JOIN pozycje poz ON poz.produkt_id = p.id;` ktoś napisał, żeby dostać listę produktów, które kiedykolwiek się sprzedały. Oceniamy, czy to jest właściwe użycie `DISTINCT`, czy plaster na inny problem.

Odpowiedź

To właściwe użycie. Złączenie z `pozycje` z natury rzeczy powtórzy produkt tyle razy, ile ma pozycji sprzedaży, a zapytanie świadomie pyta o zbiór unikalnych produktów, nie o listę transakcji — `DISTINCT` tu nie maskuje błędu w złączeniu, tylko realizuje właściwe pytanie. Alternatywą o tym samym znaczeniu byłoby `EXISTS` w podzapytaniu do `pozycje`, bez `JOIN` i bez `DISTINCT` w ogóle.

Pytania kontrolne

1. Dlaczego DISTINCT dodany do zapytania z Fiszki „Sprawdzenie przyczyny” usunął widoczne duplikaty, ale nie naprawił zapytania?

Odpowiedź

Bo problemem nie były duplikaty same w sobie, tylko niepotrzebne złączenie z tabelą podrzędną, które je wywołało — to samo złączenie wewnętrzne po cichu usunęło z wyniku zamówienia bez żadnej pozycji. `DISTINCT` scalił powtórzone wiersze, ale nie przywrócił brakujących.

2. Jaki krok w planie wykonania zdradza obecność DISTINCT w zapytaniu?

Odpowiedź

`Sort` albo `HashAggregate` (zależnie od tego, którą metodę wybrał optymalizator) — DISTINCT wymaga porównania całego wyniku ze sobą, więc silnik musi go najpierw posortować albo zahaszować.

Powiązane: SELECT: kolejność zapisu a kolejność wykonania · Czym jest złączenie: iloczyn kartezjański i warunek, który go ogranicza · Podzapytania: skalarne, wierszowe, skorelowane, EXISTS · Funkcje okna — OVER, PARTITION BY, ROW_NUMBER, RANK, LAG/LEAD

Źródła: ISO/IEC 9075-2:2016 § 7.12 · dokumentacja PostgreSQL, rozdział „SELECT DISTINCT”

Revision #47
Created 2026-09-24 22:41:53 UTC by Tomasz Konieczny
Updated 2026-10-26 14:10:56 UTC by Tomasz Konieczny